一种基于 CUDA 的截断重叠维特比译码算法

李晨杰，王志旭

(南京邮电大学 通信与信息工程学院，南京 210003)

摘　要：为解决信道译码在高吞吐量通信系统中的瓶颈问题，通过对 CUDA 并行计算的了解和对维特比译码并行实现的探索，为卷积码提出了一种基于 CUDA 的截断重叠维特比译码器。算法通过截断式的子网格图相互重叠的方式，并行执行独立的正向度量计算和回溯过程。实验结果表明，在保证了译码算法误码率性能的同时，获得了良好的吞吐量提升表现，相比现有的实现方式有 1.3~3.5 倍的提升，降低了硬件开销，能够有效运用于实际高吞吐量通信系统中。

关键词：卷积码；维特比译码；并行计算；CUDA

中图分类号：TP312　　doi: 10.3969/j.issn.1001-3695.2017.12.0794

Truncated overlap scheduling viterbi decoding algorithm based on CUDA

Li Chenjie, Wang Zhixu

(*College of Telecommunications & Information Engineering*, *Nanjing University of Posts & Telecommunications*, *Nanjing* 210003, *China*)

Abstract: In order to solve the bottleneck problem of channel decoding in high-throughput communication systems, a truncated overlap Viterbi decoder based on CUDA is proposed for convolutional codes to solve it by analyzing of parallel processing based on Compute Unified Device Architecture (CUDA) and exploring of the parallel implementation of Viterbi decoding. The algorithm performs both independent forward metrics computing and back-track procedure in parallel through the overlapping of truncated sub-grid. The experiment shows that the method keeps low BER, achieves a performance improvement of 1.3-3.5 times of the existing implementation and reduces hardware consumption. It can be effectively used in practical high-throughput communication systems.

Key words: convolutional codes; Viterbi decoder; parallel processing; CUDA

0　引言

随着现代移动通信技术的飞速发展，人们对通信业务速率的需求越来越高，给信道译码带来更高吞吐量的解决方案成为关注的重点之一。现如今，越来越多的人通过将 FPGA、DSP、GPU 等平台的优势与译码器特殊的结构相结合来达到高吞吐量的目的。文献[1,2]通过使用 FPGA 设计与实现了一种高通量的维特比译码器，文献[3]基于 DSP 实现了维特比译码算法。通过比较不难发现，DSP 虽然具有良好的代码灵活性，但是却需要依靠高主频来解决计算能力，而 FPGA 提供了高计算能力，但是开发过程非常复杂，硬件成本也非常高[4]。随着 GPU 的不断发展，它在拥有超高运算能力的同时还具有并行处理数据的能力，这给高吞吐量译码带来了更多的可能性。文献[5~7]都使用 GPU 平台达到了各自的译码器对高吞吐量需求，这说明使用 GPU 确实是一个实现高吞吐量译码器的有效途径。文献[8]提出了使用子译码级并行和共享内存技术两种方法，在 GPU 平台对 LTE Turbo 译码器进行加速，在不损失误码率的情况下有效地提高了吞吐量。文献[9]提出了基于“分治法”的维特比译码算法，通过归并的方法来消除子网格之间的相关性，这种思想非常有利于维特比译码的并行实现。

本文提出了一种基于 CUDA 的并行截断重叠维特比译码算法（Parallel-TOVDA），利用“分治法”的思想，结合子译码级并行方法和 CUDA 平台合并内存访问模式，达到了高吞吐量的要求。不同于文献[9]，本文提出的译码算法的网格图可以划分为截断式的子网格图，可以并行执行独立的正向度量计算和回溯过程。与文献[9]相比，本文提出的维特比译码器实现了 1.2~3.6 倍的性能加速和良好的误码率(BER)性能。

1　并行维特比译码算法

1.1　维特比译码算法

维特比译码过程可以表示为图 1 中描述的网格图。在网格图中，状态从左向右在多个时间阶段内反复变化，这里的时间

收稿日期：2017-12-04；**修回日期：**2018-01-30

作者简介：李晨杰（1993-），男，江苏无锡人，硕士研究生，主要研究方向为无线通信、信道编/解码（chenj371@163.com）；王志旭（1993-），男，江苏泰州人，硕士研究生，主要研究方向为无线通信、大规模 MIMO 中的导频设计与估计.

阶段数等同于信息比特的长度。维特比算法的过程可以分为正向过程和回溯过程两个方向。回溯的目的是通过网格图来搜寻最大似然路径。路径度量（PM）用来测量网格图中每种可能的路径。在每个时间阶段，计算两个状态之间的分支度量（BM），然后将它们相加来更新 PM。然而只有具有最小 PM 的路径在下一次迭代中被保留并传递下去。对于网格图中的每个状态，执行加-比-选（ACS）操作来找到所有路径的幸存比特（SB），最后得到幸存路径。ACS 操作可以如下表示：

$$let\quad temp_0 = PM_{t-1,2j} + BM_{2j,s}, temp_1 = PM_{t-1,2j+1} + BM_{2j+1,s},$$
$$s=(0,1,...,2^{K-1}),\ j = s\%2^{K-2}$$
$$if\,(temp_0 > temp_1)\quad PM_{t,s} = temp_1, SB_{t,s} = 1$$
$$else\quad = temp_0, SB_{t,s} = 0$$

其中：K 是卷积码的约束长度；分支度量 BM 等于信息比特和汉明距离的乘积。在正向过程创建整个网格图之后，回溯过程开始反向追踪以搜索出最大似然路径，该路径的所有 SB 即是最后译码得到的比特（DB）。本文以约束长度为 7，码率 1/2 的卷积码为例进行并行译码的介绍。

1.2　两种维特比译码算法的执行方式

通过对维特比解码算法的分析，可以发现在正向计算中，各状态之间的 ACS 操作相互独立，因此这些操作可以并行处理。然而从时间轴上看可以发现，$PM_{t,s}$ 是受 $PM_{t-1,s}$ 影响，因此必须逐级执行每一个时间阶段。所幸在后向计算过程中，所有的幸存者路径都是在适当的回溯长度之后就可以合并在一个状态，这种归并的过程称之为回溯阶段。最后，可以从合并的状态回溯，获得译码序列，这个过程称为译码阶段。

如图 1 所示，维特比译码算法有三个阶段，分别是正向计算阶段(FCP)、回溯阶段(TP)和译码阶段(DP)。可以发现，其实待译码的序列中的任一部分都进行上述三个阶段。如果将序列截断成 n 块，FCP 在每个块中生成子网格图，TP 得到了归并的状态，为 DP 提供起始状态，最后 DP 输出译码比特。图 2 介绍了该算法的两种执行方式：流水线式和并行同步式。文献[10]提出一种基于滑动窗口管道技术的流水线模式，箭头表示算法的执行方向。当第 i 次 FCP（记为 F_i ）完成，则第 i-1 次 TP（记为 T_{i-1} ）开始执行，结束之后再执行第 i-2 次 DP（记为 D_{i-2} ）。虽然流水线模式在使用 FPGA 时可以达到非常高的译码吞吐量[10]，但是它并不适用于 GPU。为了充分利用 GPU 上的算法单元多的特性，本文提出了基于截断重叠技术的并行同步模式。首先，输入序列被截断成 n 块，每一个块尾部都有一部分长度与下一个块重叠。重叠的尾序列将用于执行 TP 来找到归并的状态。然后将所有块中的 FCP 并行执行，在每个块中对应的 TP 和 DP 可以独立处理。

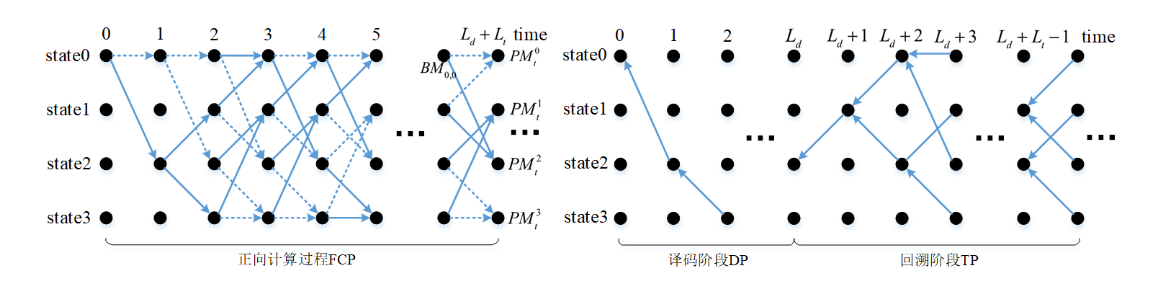


图 1　维特比译码网格图

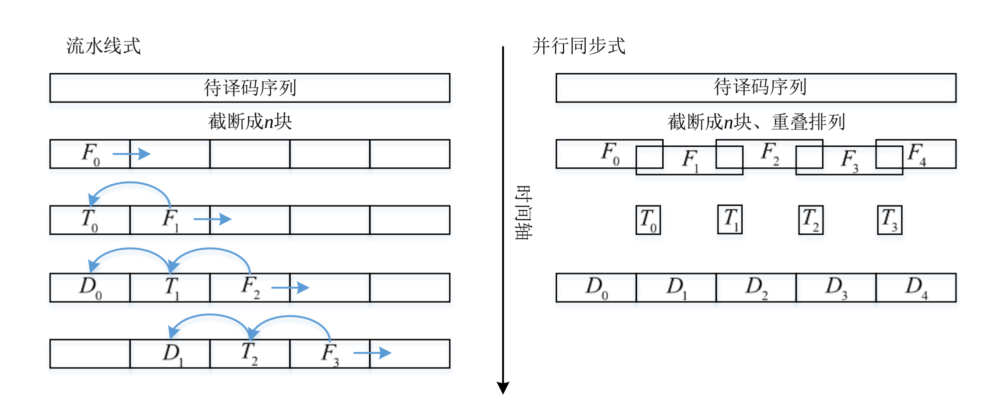


图 2　流水线式、并行同步式维特比译码

2　基于 CUDA 的 Parallel-TOVDA

2.1　基于 CUDA 平台的并行方式

CUDA 架构是一种由 NVIDIA 推出的通用并行计算架构，该架构使 GPU 能够解决复杂的计算问题，而基于截断重叠技术的维特比译码算法的思想也包含了并行计算的思想，因此只需要在该译码算法的基础上，通过对网格图的时间轴进行划分，将每个子网格图可以分为两个阶段：FCP 和回溯译码(TD)阶段，其中包括 TP 和 DP。图 3 描述了在 CUDA 上实现的并行维特比译码算法。图中 F_i 表示第 i 个截断子网格的 FCP；$ACS_{t,s}$ 表

示在时间 t 和状态 s 下的加比选操作；TD_i 表示在第 i 个截断的子网格中的回溯译码阶段。为了精准地译码，F_i 的截断长度可以设置为 $2L_d+2L_t$。其中：L_t 是 TP 的回溯长度，而 L_d 是 DP 的译码长度。这样，输入信号序列就被划分为长度为 $2L_d$ 子序列，在每个子序列尾部添加一个长度为 $2L_t$ 的尾序列，这个尾序列同时又是下一个子序列的一部分，用于执行回溯阶段。

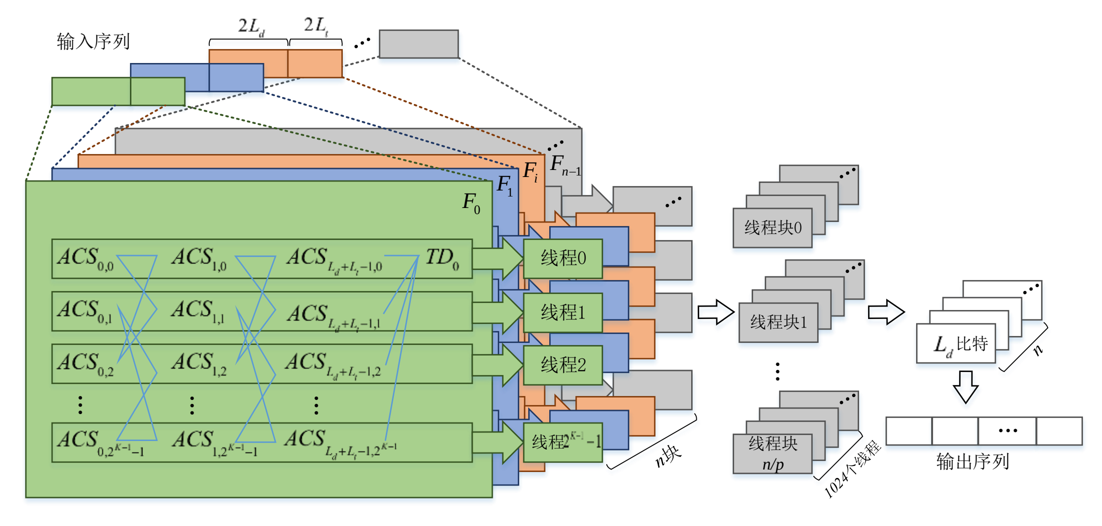


图 3 基于 CUDA 的 Parallel-TOVDA 线程安排

在 FCP 中，网格图中的一个时间阶段的每个 ACS 操作都被分配给每个线程，在其中计算 BM、PM 和 SB 值。每个线程块中总共有 1 024 个线程，一共可以进行 p 路 F_i 并行计算，p 等于 1024 除以 2K−1。连续时间阶段的网络在每个线程块中的共享内存中执行。每个线程分别将 PM 和 SB 值写入共享内存和全局内存中的对应地址。SB 值由在网格图中的所有路径组成，但这对于共享内存来说太大了。在此之后，线程间轻量级的同步可以确保所有线程在网格图的一个时间阶段中完成计算和写入操作，这并不影响性能。在同步之后，可以在共享内存中获取正确的输入 PM 值。然后在相同的情况下对各个时间阶段进行连续的计算。

在 FCP 完成后，第 i 个截断子网格中的 TD 阶段开始在 TD_i 的第一个线程中执行。采用分治法找出 F_i 中的所有第 L_d+L_t 个状态中最小的 PM 值，把它作为回溯状态。在回溯 L_t 位后就可以得到归并的状态。最后，得到的 L_d 个比特是第 i 个截断子网格的译码比特。这就是所谓的 Parallel-TOVDA。下面给出了 Parallel-TOVDA 的伪代码。

```
算法：基于 CUDA 的 Parallel-TOVDA
初始化：
输入：输入比特 c、汉明距离 hd。
输出：译码比特 DB。
变量 PMk，nextPMk 用于对 k 线程进行分配共享内存；
PMk=0；nextPMk=0；  p=16；  i=0；  BM1=0；  BM2=0；  s=0；
迭代：
for thread-block b 0 to ⌊n/p⌋ parallel do
    for thread in dimension-x tx 0 to p – 1 parallel do
        i = b × p + tx;
        for thread in y dimension ty 0 to 63 parallel do
            j = ty % 2^(K-2);
            for t 0 to Ld + Lt – 1 do
                Load c_{i+2t} and c_{i+2t+1} from the global memory;
                Load hd_{2j} and hd_{2j+1} from the constant memory;
                k = tx × 64 + ty;
                BM1 =c_{i+2t} ×hd_{2j},BM2 =c_{i+2t+1} ×hd_{2j+1};
                nextPMk = min{PM_{2j} + BM1, PM2 j+1 + BM2};
                SB_{t,ty} = (PM_{2j} + BM1 > PM_{2j+1} + BM2)?1 : 0;
                Store S B_{t,ty} in the global memory;
                Synchronize;
                PMk = nextPMk;
            end for
        end for
        Reduce and Fetch the state with minimum PM s_min
from PMα+Ld+Lt,0 to PMα+Ld+Lt,63;
        s = s_min;
        for t Ld + Lt – 1 to 0 do
            j = s%2^(K-2);
            Load S B_{t,s} in the global memory;
            if t<Ld  then
                DBt = SB_{t,s};
            end if
            s = ( SB_{t,s} = 0 ) ? 2j : 2j+1;
        end for
    end for
end for
```

2.2 合并内存访问技术

在算法中，每个线程中的 ACS 操作需要在每次迭代中将 SB 存储到全局内存中，从而产生大量的内存访问开销。但是通过合并内存访问模式，可以将内存访问的总时间减少 16 倍。在

CUDA 中，*half-warp* 作为调度单位，内部拥有 16 个线程，它可以启动一个统一的合并访问内存，而不是启动 16 个单独的内存访问。这种合并访问模式只在半经线的数据必须存储在连续的内存地址中的情况下才可以使用。因此，必须在全局内存中对 SB 组织进行调整，以执行合并的内存访问。在编译之前，SB 必须以一种连续的方式对一个线程进行处理，以便在 64 个存储在一起的状态中形成 SB。然后，在一个 *half-warp*，即每 16 个线程一组，可以通过单一的连接内存访问，在全局内存中执行数据交换。对于这种 CUDA 特有的内存访问技术，这里不再赘述。

3　误码率性能及吞吐量

3.1　性能分析模型

处理译码的 GPU 频率为 F，流处理器数量为 N_{CUDA}。算法中用 N_s 表示状态数，n 是分块数，执行周期和内存访问延迟在 ACS 操作时分别为 E_{ACS} 和 M_{ACS}，在回溯操作时分别为 E_{TD} 和 M_{TD}。因为 FCP 中合并内存访问，所以 M_{ACS} 需要除以 16。由于在 TP 阶段加入了额外的长度为 L_t 的比特，所以在回溯操作 E_{TD} 和 M_{TD} 中执行和内存访问循环包括 TP 和 DP 的部分。该算法的执行时间 T_p 为

$$T_p = \frac{n}{F \times N_{CUDA}}\left[\left(E_{ACS} + \frac{M_{ACS}}{16}\right) \times N_s + \left(E_{TD} + M_{TD}\right) \times \left(L_d + L_t\right)\right]$$

3.2　实验设置

为了验证基于 CUDA 的 Parallel-TOVDA 的实用性，本文给出的测试场景是：输入随机生成的二进制比特流，由卷积编码器编码，然后经过 AWGN 通道。在接收机中，数据传递到 GPU 的全局内存中，执行 Viterbi 解码过程。

在测试场景中，GPU 为 GTX1050，拥有 640 个 CUDA 核心，核心的基础频率是 1.35 GHz，可动态睿频至 1.5 GHz。其全局内存的大小是 2 GB。表 1 列举了本次实验的关键参数设置情况。

表 1　测试实验关键参数设置情况

	误码率性能测试实验	吞吐量性能测试实验
总译码位数	16384	16384
L_t	0、14、28	14
L_d	64	32、64、128、256、512
N_b	—	1、2、4、8、16、32

3.3　译码器误码性能

如前所述，L_t 表示了 TP 的长度，它可以决定解码过程的初始状态并影响维特比译码器的误码率。为了评估译码器的 BER 性能，将 L_t 长度分别设置为 0、14、28，固定 L_d 长度为 64。最终性能指标如图 4 所示。BER 随 L_t 的增加而减小。当回溯长度越高，译码器的 BER 在 MATLAB 仿真中越接近理论 BER 的性能，这证明了本文的译码器是实用的。

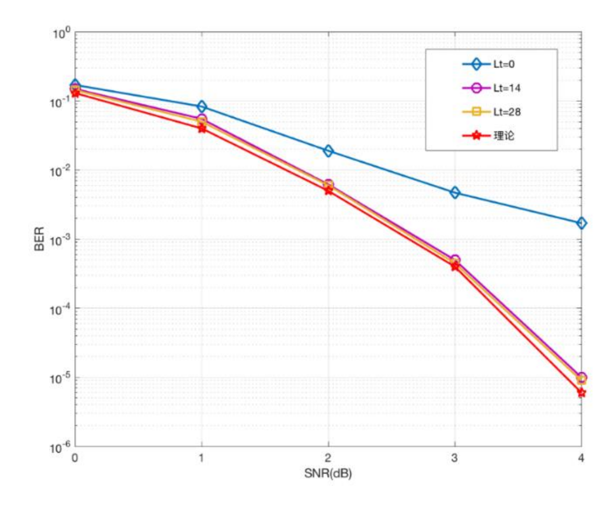


图 4　AWGN 信道下译码器在不同 L_t 情况的误码率性能

3.4　译码器吞吐量

为了测量在 CUDA 上提出的维特比译码算法的处理时间，本文使用 NVIDIA 公司提供的 Profiler。在实验中，总译码位数是 16 384，被分成 n 个截断块，在 GPU 的 N_b 个线程块上执行，因此固定 L_t 为 14，改变不同的 N_b 和 L_d。如图 7 所示，发现开启线程块越多，执行的效率越高，吞吐量越大；当运行在相同的线程块情况下，L_d 设为 64 可以获得更好的执行效率，吞吐量相对较高。纵观图 5，当 L_d 为 64 并开启 8 个线程块并行执行时（白线交点），CUDA 核心的计算能力和线程块间共享内存的使用达到了平衡。

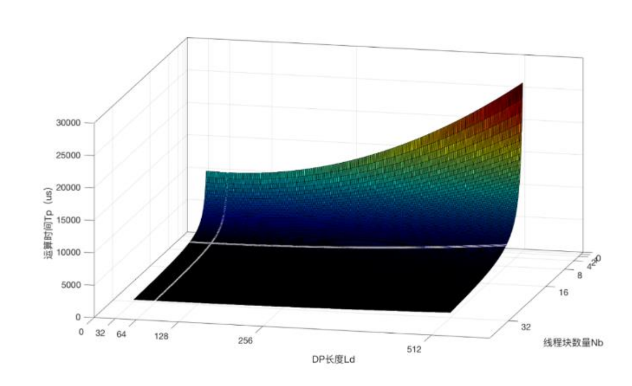


图 5　不同 DP 长度和线程块情况下的运算时间

3.5　性能比较

为了突显 Parallel-TOVDA 的优点，本文将实验结果与文献[9,11]进行对比（表 2）。文献[9]中的算法利用了将比特序列分割成多个块的方法，将这些块被分布到与本文相似的多个线程块中。然而在回溯阶段，多个线程块的所有路径都被合并，并且通过一个线程生成解码序列。本文提出的算法是一种完全并行的截断重叠维特比算法。整个网格图被分割成多个子网格图，每个子网格图的正向计算和回溯独立执行。性能比较如图 8 所示。该维特比译码算法的峰值吞吐量达到了 88.1 Mbps，总体相比文献[9]吞吐量提高了 1.3~3.5 倍。文献[12~14]通过提高 CUDA 核心的频率和数目，在原有实现方式的基础上获得了吞

吐量的提升，甚至到 Gb/s 级别。

表 2 不同线程块 N_b 时译码器吞吐量比较/Mbps

N_b	1	2	4	8	16	32
文献[11]	0.9	1.8	4.1	7.6	14.8	28.7
文献[9]	2.7	4.7	8.6	12.3	15.1	29.4
本文	3.5	7.1	13.4	28.3	52.9	88.1

图 6 更直观地显示了本文所设计的译码器在吞吐量上的巨大优势。性能优化情况如下：当线程块 N_b 设置为 16 时，Parallel-TOVDA 相比文献[9]所提出的译码器在吞吐量上提高了 3.5 倍，相比文献[11]提高了 3.6 倍；当达到 88.1Mbps 峰值吞吐量时，相比文献[9]和[11]提高了 3 倍；当仅试用 1 个线程块（N_b =1）时，相比文献[9]提升了 1.3 倍。这说明在该算法在保证纠错性能前提下，最大限度地将维特比译码的结构特性和 CUDA 平台的计算特性结合，提高了译码的吞吐量，这意味着该算法可以在同样的吞吐量要求下，节省更多的硬件开销。

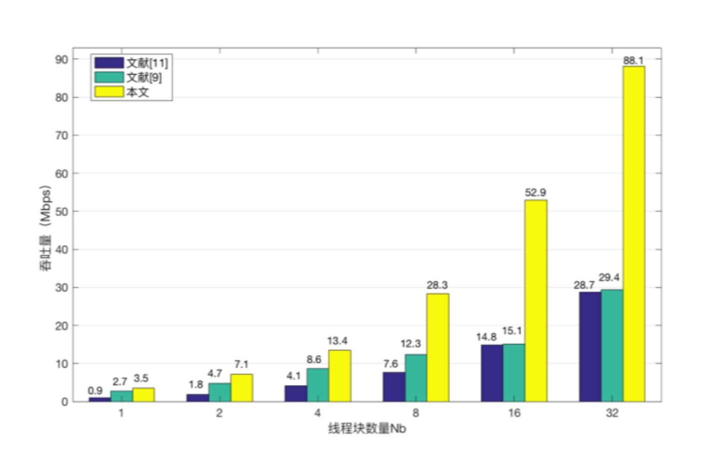


图 6 不同线程块 N_b 时译码器的吞吐量大小/Mbps

4 结束语

本文提出了一种基于 CUDA 的截断重叠并行维特比译码算法。将维特比译码器的输入比特序列划分为 *n* 个重叠的截断块，分布在 GPU 上的多个进程块中，每个线程块可以对多个截断块进行并行处理。在每个截断块中，正向计算和回溯可以独立执行，从而最大限度地提高 GPU 的并行能力。实验表明，基于 CUDA 的截断重叠维特比译码器的吞吐量可以达到 88.1 Mbps，相对于文献[9]的实现有 1.3~3.5 倍的性能改进。如果对吞吐量有更高的要求，可以使用 CUDA 核心数目更多、频率更高的 GPU 平台，线性地提高吞吐量，很轻易就可以达到 Gb/s 级别的吞吐量。结果表明，提出的维特比译码器完美匹配 CUDA 平台的架构特性，能够在保证纠错能力的同时很大程度上提高了译码吞吐量，可用于基站或移动基站的通信。

参考文献：

[1] Kermani M M, Singh V, Azarderakhsh R. Reliable low-latency viterbi algorithm architectures benchmarked on ASIC and FPGA [J]. IEEE Trans on Circuits & Systems I Regular Papers, 2017, PP (99): 1-9.

[2] Li L F, Li H W, Li H L, *et al.* Research and implementation of Viterbi decoding in TD-LTE system [C]// Advanced Information Technology, Electronic and Automation Control Conference. 2017: 890-894.

[3] 罗友宝，李小文. LTE 系统的 Viterbi 译码算法仿真及 DSP 实现 [J]. 光通信研究, 2010, 2010 (3): 67-70.

[4] Stamoulias I, Georgoulakis K, Blionas S, *et al.* FPGA implementation of an MLSE equalizer in 10 Gbps optical links [C]// Proc of IEEE International Conference on Digital Signal Processing. 2015: 794-798.

[5] Wu M, Wang G, Cavallaro J R. Implementation of a high throughput 3GPP turbo decoder on GPU [J]. Journal of Signal Processing Systems, 2011, 65 (2): 171-183.

[6] Martínez-Zaldívar F J. Tridimensional block multiword LDPC decoding on GPUs [J]. Journal of Supercomputing, 2011, 58 (3): 314-322.

[7] Wu M, Sun Y, Gupta S, *et al.* Implementation of a high throughput soft MIMO detector on GPU [J]. Journal of Signal Processing Systems, 2011, 64 (1): 123-136.

[8] Liu C, Bie Z, Chen C, *et al.* A parallel LTE turbo decoder on GPU [C]// Proc of IEEE International Conference on Communication Technology. 2014: 609-614.

[9] Lin C S, Liu W L, Yeh W T, *et al.* A tiling-scheme viterbi decoder in software defined radio for GPUs [C]// Proc of International Conference on Wireless Communications, Networking and Mobile Computing. 2011: 1-4.

[10] Zhang D, Zhao R, Han L, *et al.* An implementation of viterbi algorithm on GPU [C]// Proc of International Conference on Information Science and Engineering. 2009: 121-124.

[11] Ahn C, Kim J, Ju J, *et al.* Implementation of an SDR platform using GPU and its application to a 2 × 2 MIMO WiMAX system [J]. Analog Integrated Circuits & Signal Processing, 2011, 69 (2-3): 107-117.

[12] Li R, Dou Y, Zou D. Efficient parallel implementation of three-point viterbi decoding algorithm on CPU, GPU, and FPGA [J]. Concurrency & Computation Practice & Experience, 2014, 26 (3): 821-840.

[13] Peng H, Liu R, Hou Y, *et al.* A Gb//s parallel block-based Viterbi decoder for convolutional codes on GPU [C]// Proc of International Conference on Wireless Communications & Signal Processing. 2016: 1-6.

[14] Xia K F, Bin W U, Xiong T, *et al.* Design of a high-throughput sliding block viterbi decoder for IEEE 802. 11ac WLAN systems [J]. Ieice Trans on Fundamentals of Electronics Communications & Computer Sciences, 2017, E100. A (8): 1606-1614.